第17回

形式的検証ツールの効果的な利用法(2)

——RTL記述どうしの等価性を検証する

藤田昌宏

今回は，RTL(register transfer level)記述どうしの形式的検証について解説する．とくに，タイミングを調整するために回路を人手により修正した(いわゆるリタイミング)前後の等価性検証について詳しく述べる．シミュレーションの場合と異なり，形式的検証にはコーナ・ケースを確実に検出できるという利点がある．ただし，回路構成によって組み合わせ回路向けの等価性検証ツールと順序回路向けの等価性検証ツールを適切に使い分ける必要がある．（編集部）

前回は，Cベースの言語で表現された設計記述どうしの等価性を，市販の等価性検証ツールを使って検証してみました．検証対象としては，アルゴリズムの表現として書かれたSystemC記述から出発し，記述の最適化や並列性の導入，パイプライン化を人手で行った設計との等価性を確認しました．

今回はその続きです．RTLの設計記述に対する最適化として，おもにリタイミングによる最適化前後の検証における等価性検証ツールの適用，およびその性能について述べたいと思います．また，等価性検証ツールの利用に関する一般的な注意点もまとめます．

● 等価性検証問題は三つのケースに分類できる

RTLの設計記述は，基本的に同期回路で構成した順序回路を表現しており，各クロックにおける動作が完全に規定されています．つまり，各クロックごとに，どの変数(レジスタやメモリ)の値を利用して，どのような演算を施し，結果をどの変数に格納するかが記述されています．このような二つのRTLの設計記述の間の等価性検証問題は，次の三つに分類できます．

1) 各クロックの個々の動作は異なるが，順序回路全体の動作として等価か否かを判定する問題

2) 各クロックの個々の動作は同じだが，フリップフロップの対応が取れていない場合の順序回路全体の等価性を判定する問題

3) 各クロックの個々の動作が同じで，かつフリップフロップの対応が取れている場合の等価性を判定する問題

3)は，現在広く利用されている組み合わせ回路向けの等価性検証ツールをそのまま利用できる場合です．フリップフロップの対応は1対1になる場合もありますが，回路が冗長だと，片方の設計の一つのフリップフロップがもう一方の複数のフリップフロップに対応する場合があります．この対応は，フリップフロップの名まえや回路のトポロジーとしての構造から判断したり，設計者が指定したりします．多くの検証ツールには，こうしたフリップフロップの対応を自動的に見つける機能が備わっています．比較している二つの設計記述に，回路の中の組み合わせ回路の部分を単純に等価変換した程度の差しかない場合，これは**3)**のケースとなり，基本的に組み合わせ回路部分だけの等価性を調べればよいことになります．結果として，多くの場合，数百万～数千万ゲート規模の設計でも数時間程度で等価性を検証できます．形式的検証ツールが完全に実用的に，また日常的に利用されている数少ないケースの一つです．

一方，フリップフロップの対応が完全には取れないときは**2)**になります．ここで「対応が取れない」というのは，

KeyWord 形式的検証，等価性検証，コーナ・ケース，SystemC，リグレッション・テスト，パイプライン，リタイミング，ワンホット・エンコーディング，マイクロアーキテクチャ，モデル・チェッキング，ドント・ケア

「ほんとうは対応が取れるはずだが，うまく見つけられない」ということです．本質的に対応が取れないのは**1)**の場合です．**1)**と**2)**は異なるのですが，等価性検証ツールから見ると両者は部分的にしかフリップフロップの対応がとれない(あるいは，まったく対応が取れない)場合であり，基本的に扱いは同じになります．つまり，組み合わせ回路の部分だけを解析するのでは不十分であり，順序回路として時間シーケンス上の動作を解析する必要があります．結果として，扱える回路規模は大幅に小さくなります．**3)**の場合のように，数百万ゲート以上の設計を扱うことは不可能で，その1/10以下の，数万～数十万ゲート規模が限界です．一度に扱える回路規模は一つの回路ブロック程度であり，LSI設計における論理合成やビヘイビア合成(高位合成)を行える規模と一致しています．

なお，以上の議論は二つの比較している設計が等価である場合です．等価ではないときは，基本的にフリップフロップの対応は取れないはずです．逆に，フリップフロップの対応がまちがって認識されていると，等価である設計も不等価であると判定されてしまいます．つまり，不等価と判定された場合には，ほんとうに不等価の場合と，フリップフロップの対応をまちがえている場合の2通りが考えられます．これらのどちらであるかを解析するには，不等価と判定された場合にツールから生成される反例を人手で解析するか，あるいは**1)**の場合であるとして(つまり，フリップフロップの対応は考えないようにして)，等価性検証をやり直すなどが考えられます．しかし後者は，上でも述べたように大規模設計には適用できません．

設計記述1

```
out=a+b;
```

設計記述2

```
out=a-b;
```

(**a**) 判定が簡単な場合

設計記述3

```
if(x=12345)out=a+b;
```

設計記述4

```
if(x=12346)out=a-b;
```

(**b**) 判定が簡単ではない場合

図1 コーナ・ケースの例

(a)のa＋bとa－bの値が異なる場合は多数あり，シミュレーションの中でaとbの値をランダムに選んでも，結果のoutが異なる可能性は高い．(b)の二つの記述において，出力outの値が異なるのはxの値が12345か12346になった場合だけ．xが32ビット幅の変数であるとすると，その値は$2^{32}-1$通りありえることになり，ランダムに値を選んでxの値が上の二つになる可能性はきわめてゼロに近い

● コーナ・ケースを確実に検出できる

一般に，二つの設計が不等価であることはシミュレーションでも検出可能なはずですが，これは反例が多数存在する場合に限られることに注意してください．例えば，**図1(a)**の二つの設計記述が不等価であることはランダム・シミュレーションなどで簡単に示すことができます．

図1(a)のa＋bとa－bの値が異なる場合は多数あり，シミュレーションの中でaとbの値をランダムに選んでも，結果のoutが異なる可能性は高いといえます．これに対して，少し複雑にした**図1(b)**の二つの設計記述が不等価であることをランダム・シミュレーションによって判定しようとすると，簡単ではありません．**図1(b)**の「設計記述4」は，本来「x＝12345」とするべきところをタイプ・ミスか何かで「x＝12346」とまちがえてしまった場合に相当します．この二つの記述において，出力outの値が異なるのはxの値が12345か12346になった場合だけです．一方，xが32ビット幅の変数であるとすると，その値は$2^{32}-1$通りありえることになり，ランダムに値を選んでxの値が上の二つになる可能性はきわめてゼロに近いと言えます．つまり，ランダム・シミュレーションでは不等価であることを発見できないということです．

このように，不等価であることを示す反例の数がきわめて少ない場合に検証が難しくなることを「コーナ・ケース問題」と呼びます．このコーナ・ケース問題があるため，シミュレーションによる検証では，通常，非常に多数のシミュレーション・パターンが利用されます(いわゆるリグレッション・テスト)．これらのパターンの生成は，上記の例のようなことも考えながら，ある程度人手で行う必要があり，作成にもけっこう時間がかかります．このことから，形式的な等価性検証の重要性は，強く認識されています．

例えば，1回のシミュレーションによるリグレッション・テストに，準備も含めて15日かかるものとします．これに対して形式的な等価性検証ツールでは，準備も含めて3日かかるとしましょう．形式的な等価性検証ツールは，後述するように一つの回路ブロックの検証に数分～数時間程度かかりますが，準備も含めていちおう3日とします．1年間に10種類の設計を行っており，一つの設計は平均10の回路ブロックがあるとします．その回路ブロックのうち，回路規模や回路の性質から，半分が形式的検証ツールで等価性判定が可能であるとします．さらに，一つの回路ブロック当たり平均10回程度の設計最適化が施され，結果とし

て等価性を検証する必要があるとします．すると，形式的な等価性検証ツールを導入することによる設計期間の短縮（工数削減）は以下のようになります．

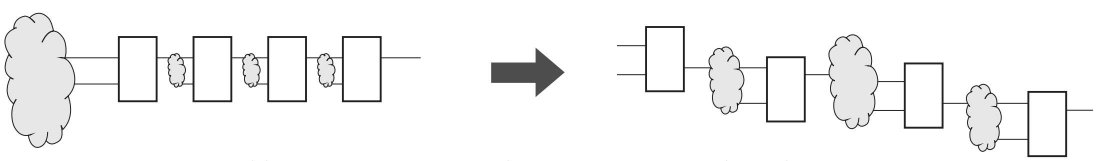
10 × 10 × 0.5 × 10 ×（15 − 3）＝ 6,000 人日

つまり，1年当たり6,000人日分の工数削減の効果が期待できることになります．これはかなり大きな効率化の効果と言えます．現状では，このように完全にはリグレッション・テストを行わず，結果として時間切れでテープアウト，出荷ということも多いと思います．出荷しているものにバグがなければ現状の方法でもよいことになりますが，実際にはそううまくいかないようです．

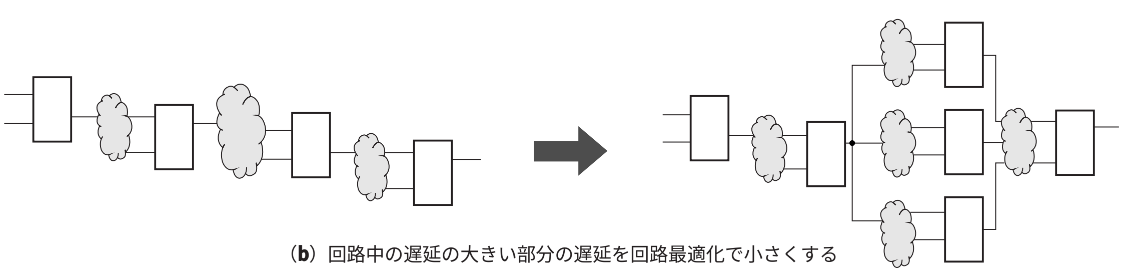
一つの回路ブロック当たりの設計最適化による等価性検証の回数が10回という仮定は多いようにも思えますが，回路の高性能化のためのリタイミングやタイミング収束問題回避のための回路変更などは，かなりの回数行われていると思います．とくに高性能なマイクロプロセッサなどの設計では，パイプライン処理の変更や，カウンタ・ベースの状態割り当てからワンホット・エンコーディングへの変更，さらに部分的にラッチ・ベースの設計を導入するなど，一度論理合成した設計に対してさまざまな変更が加えられています．また，設計を始めた当初は，設計の正しさやわかりやすさの観点から“きれいな”RTLの設計記述を作成するのですが，プロジェクトの中盤以降は，性能向上の観点から大幅に記述を修正することも多いと思います．このような点からも，形式的な等価性検証ツール導入の効果は高いと言えます．

● 組み合わせ回路ブロックの一部を移すリタイミング

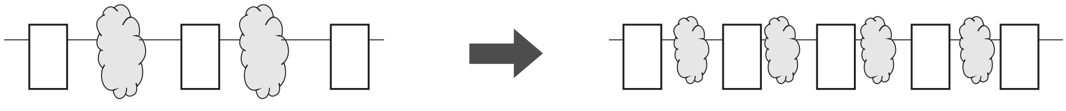
RTL設計の最適化としてもっとも頻繁に行われるのが，リタイミング処理です．同期式順序回路では，回路遅延のもっとも大きい部分によってクロック周期が決定されます．そこで，リタイミングでは，回路中の各組み合わせ回路部分の遅延の不均一を可能な限り取り除き，結果としてより速いクロックで動作する回路を実現します．具体的には，遅延時間の大きい組み合わせ回路ブロックの一部の回路を遅延時間の小さい組み合わせ回路ブロックに移すのですが，このときフリップフロップをまたいで移動させることになります．したがって，組み合わせ回路ブロックとしては同じ論理関数を実現しておらず，もはや等価ではありません．



（a）回路中の組み合わせ回路ブロックの遅延をバランス（均等化）させる



（b）回路中の遅延の大きい部分の遅延を回路最適化で小さくする



（c）パイプラインの段数を増やすことで遅延の縮小を図る

図2　典型的なリタイミングの例

（a）は，回路中の組み合わせ回路の遅延のばらつきが大きい場合．それらをバランスさせるためにフリップフロップをまたいで組み合わせ回路を部分的に移動させている．（b）は，（a）のバランス化に加えて，特定の組み合わせ回路部分の遅延が大きいので，そこを三つに分けて並列に計算することで遅延の縮小を図っている．（c）では，組み合わせ回路の遅延を小さくするためにパイプライン・ステージの数を増やして，各ステージの遅延を減らしている

しかし，単に組み合わせ回路ブロックの一部を移動させただけなので，順序回路としての動作は（リセット状態からの数クロックの動作を除き）同じになります．上流の設計工程では遅延を各組み合わせ回路ブロックに十分均等に配分したつもりでも，論理合成やレイアウト設計のつごうにより最終的な遅延が変化することはよくあります．リタイミング処理は，高性能LSIの設計ではごく日常的に行われています．

さて，このようなリタイミングされた設計の間の等価性はどのように検証することになるのでしょう．問題の分類では，前述の**1)**の場合になります．すなわち，順序回路としての等価性を検証するツールを利用すれば，判定できます．しかし，扱える回路規模や検証速度の観点から，組み合わせ回路向けの等価性検証ツールが利用できるならば，それにこしたことはありません．実際，リタイミングは組み合わせ回路ブロックを部分的に移動させただけです．そこで，どのような場合に組み合わせ回路向けの等価性検証ツールが利用できて，どのような場合には利用できないかについて見ていきましょう．

● フリップフロップを除けば組み合わせ回路用で検証可能

典型的なリタイミングの例を**図2**に示します．**図2(a)**は，回路中の組み合わせ回路の遅延のばらつきが大きい場合に，それらをバランス（均等化）させるためにフリップフロップをまたいで組み合わせ回路を部分的に移動させています．**図2(b)**は，**図2(a)**のバランス化に加えて，特定の組み合わせ回路部分の遅延が大きいので，そこを三つに分けて並列に計算することで遅延の縮小を図っています．三つに分けているので，フリップフロップの数も増大しています．**図2(c)**では，組み合わせ回路の遅延を小さくするためにパイプライン・ステージの数を増やし，各ステージの遅延を減らしています．このようなパイプライン段数の変更も高性能LSIの設計では頻繁に行われています．

図2の例では，いずれの場合も回路全体としてループがなく，処理は左から右に進んでいくようになっています．しかし，実際には，部分的に回路のループがある場合にリタイミングが適用されることも多いようです．回路としてループがあると，扱いが複雑になります．**図2**の例のように回路全体としてループがない場合には，組み合わせ回路向けの等価性検証を適用できます．

基本的にはフリップフロップを取ってしまって，フリップフロップの入出力をそのまま接続すると，一つの大きな組み合わせ回路となります．それらの間で等価性検証を行えばよいわけです．あるいは，フリップフロップをリタイミングの要領で回路の出力側（**図2**の例では右側）へすべて移動させると，フリップフロップの左側は完全な組み合わせ回路となり，組み合わせ回路向けの等価性検証ツールで扱えるようになります．

このように比較的単純なリタイミングは，組み合わせ回路向けの等価性検証ツールで扱える場合が多くあります．例えば，**図2**に示す例はすべて扱えます．

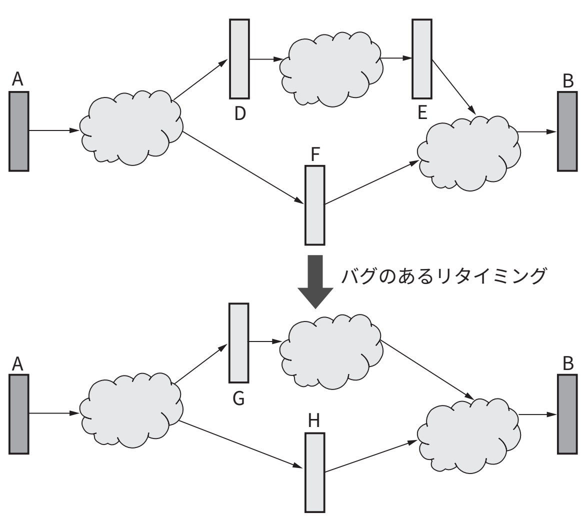


図3　フリップフロップやラッチを取り除く変換では対応できない例
単純にフリップフロップを取り去るような変換を行って組み合わせ回路として等価性を検証しようとすると，等価であると判定される．しかし実際には，上の回路のA，D，E，Bというパスと，下の回路のA，G，Bというパスにおいて横切るフリップフロップの数が異なる場合，不等価になる

● ループや再収れん，リセットは一筋縄ではいかない

一方，回路中にループが部分的にでもある場合には，上記のようなフリップフロップをすべて出力側に移動させるということはできなくなり，組み合わせ回路向けの等価性検証ツールでは扱えません．

また，回路に再収れんがあるような場合も注意が必要です．**図3**に示すようなリタイミングを考えます．これは誤ったリタイミングの例で，フリップフロップD，E，FがフリップフロップG，Hに変換されています．これを単純にフリップフロップを取り去るような変換を行って組み合わせ回路として等価性を検証しようとすると，等価であると判定されてしまいます．しかし実際には，上の回路のA，D，E，Bというパスと，下の回路のA，G，Bというパス

において横切るフリップフロップの数が異なる場合，不等価になります．したがって，このような再収れんのある回路では，単純な変換によって等価性検証をうまく行うことができません．

さらに，回路変換を行い，リタイミングをキャンセルして組み合わせ回路向けの等価性検証ツールを適用する場合は，回路のリセット状態にも注意を払う必要があります．**図4**の例を見てください．上の回路は下の回路のフリップフロップA，B，C，DをANDゲートとNANDゲートの右側に移動させて，フリップフロップEとしたものです．この回路にはループなどはありませんから，組み合わせ回路向けの等価性検証ツールを適用できそうに見えます．実際，上記のような方法でリタイミングの逆変換を行うと二つの回路は同一になり，組み合わせ回路向けの等価性検証ツールで等価と判定されます．

しかし，ここで問題があります．二つの回路において，すべてのフリップフロップでリセット値(初期値)が‘0’であるものとします．この場合，実は二つの回路の動作は異なります．上の回路のフリップフロップEのリセット値は‘0’であり，回路はそこが‘0’として動作を開始します．これに対して下の回路では，フリップフロップA，B，C，Dのリセット値が‘0’です．すると，下の回路において，上の回路のフリップフロップEのある部分に相当するNANDゲートの出力の値は‘1’になります．この部分に相当する値は上の回路では‘0’であるため，この部分以降の動作は二つの回路で異なることになります．

このように，回路のリセット値はリタイミング変換に応じて適切に修正する必要があります．これをまちがえた誤設計は，単純に組み合わせ回路向けの等価性検証ツールを適用しても不ぐあいを検出することができません．とくに**図4**の例は誤った設計を正しいと判定してしまう場合なので，注意が必要です．

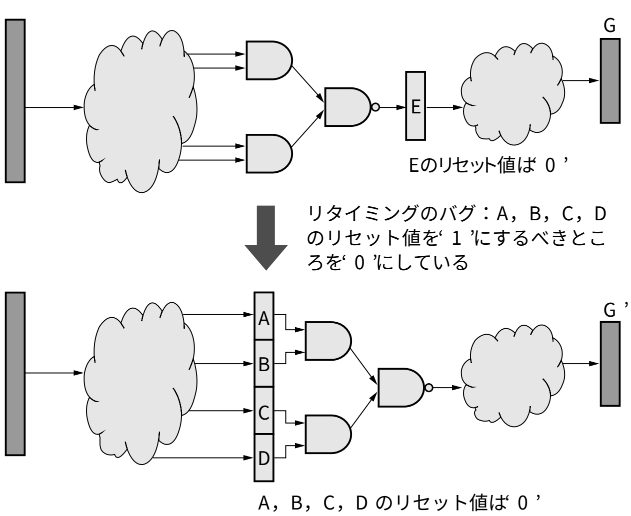
これを避ける一つの方法は，回路中のフリップフロップのリセット値がリタイミング変換の前後で正しく保たれているかをチェックすることです．しかし，人手でこれを行うには限界があります．そのような場合は，順序回路向けの等価性検証ツールを利用するしか手はありません．

● マイクロアーキテクチャが異なる場合はくふうが必要

単なるリタイミングではなく，マイクロアーキテクチャそのものが異なる場合でも，組み合わせ回路向けの等価性検証ツールを利用できるケースがあります．例えば，制御回路部分の状態割り当てのみが異なるような二つの設計の等価性について，異なる状態割り当ての間の対応をなんらかの方法で組み合わせ回路向けの等価性検証ツールに知らせることができれば，検証可能な場合があります．

片方がワンホット・エンコーディングで，もう一方が一般的な状態割り当てになっている場合，両者の状態割り当てを互いに変換し合う回路を自分で作り，それらを通して二つの回路のフリップフロップを接続することで，基本的に等価な組み合わせ回路に変換できます．ただしこの方法では，状態割り当ての変換を行う回路を自分で作る必要があり，それが正しくないと等価性検証の結果が誤ってしまうことになります．また，状態が割り当てられていないフリップフロップのパターンをドント・ケア(dont care)として利用する最適化が施されている場合，具体的にどのようなドント・ケアを回路最適化の際に利用したかが不明だと，等価性の判定を行うことはできません．

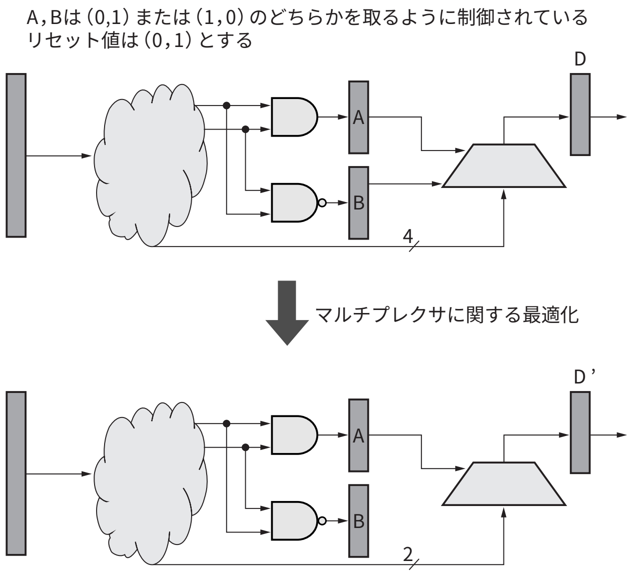
こうした「使われないフリップフロップの値のパターン」を利用した回路最適化は，よく行われます．**図5**を見てください．上の回路では，左の回路ブロックでフリップフロ



●リタイミングの逆変換でA，B，C，DからEを生成する
●しかし，組み合わせ回路向けの等価性検証ツールでは，各フリップフロップのリセット値を考慮できない

図4　リタイミングの逆変換による検証における問題点

二つの回路において，すべてのフリップフロップでリセット値(初期値)が‘0’の場合，二つの回路の動作は異なる．上の回路のフリップフロップEのリセット値は‘0’であり，回路はそこが‘0’として動作を開始する．これに対して下の回路では，フリップフロップA，B，C，Dのリセット値が‘0’である．すると，下の回路において，上の回路のフリップフロップEのある部分に相当するNANDゲートの出力の値は‘1’になる．この部分に相当する値は上の回路では‘0’であるため，この部分以降の動作は二つの回路で異なることになる



●組み合わせ回路向けの等価性検証ツールでは，A，Bの値として任意の値を取ると仮定するため，不等価と判定する
●順序回路向けの等価性検証ツールでは，A，Bの値として取りうるものをリセット値から順に計算していくため，（0，1）または（1，0）しか取らないと判定でき，等価であることを証明できる

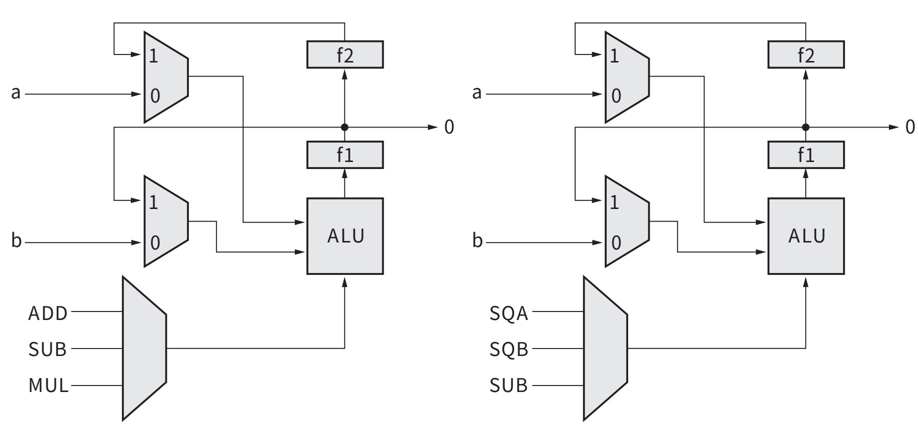
図5　マイクロアーキテクチャが異なる最適化の例

上の回路では，左の回路ブロックでフリップフロップA，Bの値を計算し，その4通りの値に応じてマルチプレクサの入力が選択されている．下の回路では，それがフリップフロップAの値の2通りのみから選択されるようになっている．しかし実際には，フリップフロップA，Bの値の組み合わせは（1，0）または（0，1）の2通りしかない

ップA，Bの値を計算し，その4通りの値に応じてマルチプレクサの入力が選択されています．下の回路では，それがフリップフロップAの値の2通りのみから選択されるようになっています．しかし回路を見ると，実際には，フリップフロップA，Bの値の組み合わせは（1，0）または（0，1）の2通りしかありません．これは，同じ入力につながっているANDゲートとNANDゲートの出力がフリップフロップA，Bに接続されていることから容易に解析できます．

ただし，組み合わせ回路向けの等価性検証ツールでは各フリップフロップの部分で回路を分けて，それぞれの組み合わせ回路部分の等価性を検証します．そのため，**図5**の例では二つの回路の間で組み合わせ回路の対応がとれず，等価性検証を行うことはできません．一方，順序回路向けの等価性検証ツールでは，フリップフロップA，Bの取りうる値の集合を計算し，それらを考慮して等価性判定を行います．すなわち，**図5**の二つの回路の等価性を正しく検証できます．

計算アルゴリズムを修正するような回路最適化も，そのままでは組み合わせ回路向けの等価性検証ツールで検証できません．**図6（a）**の回路では，（a＋b）×（a－b）として計算しています．また，**図6（b）**の回路では，それをa^2-b^2として計算しています．これらは等価な算術式であり，その意味ではこの二つの回路は等価です．しかし，**図6**に示すように，順序回路として計算式を3サイクルかけて計算しているため，組み合わせ回路としては等価ではありません．二つの回路はともに3サイクルごとに同じ計算結果が得られる回路になっています．ここで，もし二つの順序回路をともに3サイクル分，時間軸上に展開するような回



サイクル	演算
1	a+b
2	a−b
3	f2×f1

（**a**）（a+b）×（a−b）

サイクル	演算
1	a^2
2	b^2
3	f2−f1

（**b**）a^2-b^2

図6
計算アルゴリズムを修正する最適化の例

（a）の回路では，（a＋b）×（a－b）として計算している．（b）の回路では，それをa^2-b^2として計算している．これらは等価な算術式であり，その意味ではこの二つの回路は等価．しかし，順序回路として計算式を3サイクルかけて計算しているため，組み合わせ回路としては等価ではない

路に変換できる場合は，その展開された回路どうしが組み合わせ回路として等価になるため，組み合わせ回路向けの等価性検証ツールを利用できます．

このように，一般にマイクロアーキテクチャを変更するような回路最適化の場合，さまざまなくふうを行わない限り，順序回路向けの等価性検証ツールを利用する必要があります．

● 一定サイクル内の等価性しか保証できないことがある

順序回路向けの等価性検証ツールの実設計への適用を考えてみると，RTLの設計記述どうしの等価性検証については，設計全体ではなく，設計中の各ブロックに適用するのが一般的です．このとき現実的に検証可能なのは，回路規模としては数万～数十万ゲート，数百～数千フリップフロップ以内の場合です．この場合，検証時間は，数十分～数時間程度となります．ただし，設計によっては完全な検証ではなく，一定サイクル数以内の等価性しか検証できないこともよくあります．これは，一般に「バウンデッド（bounded）等価性検証」と呼ばれるものです．

ほんとうの等価性検証は，任意のサイクルの入力に対して等価であることを証明することですが，これを行うには検証ツールの中でなんらかの数学的帰納法の証明を行うか，あるいは回路を順次調べていき，すべての到達可能な状態を調べつくす処理が必要になります．これは基本的にモデル・チェッキングと同じ処理です．

モデル・チェッキングにもバウンデッド・モデル・チェッキングがあるように，等価性検証にもバウンデッド手法があります．バウンデッド等価性検証では，一定サイクル数内の等価性のみが保証されます．したがって，バウンデッド等価性検証で不等価と判定された場合はほんとうに不等価ですが，等価と判定された場合はほんとうに等価かどうかわかりません．このようなケースでは割り切って，順序回路向けの等価性検証ツールを利用する必要があります．

● ドント・ケアを最適化に利用した場合は条件指定が必要

最後に，順序回路向けの等価性検証ツールを実際に使ううえでの注意点を一つ述べたいと思います．

順序回路向けの等価性検証ツールをLSI全体の設計にそのまま適用することは，回路規模の点から不可能です．そこで，各回路ブロックごとに等価性を順に検証していくことになります．各回路ブロックはそれぞれ独立に動作しているわけではなく，お互いに通信しながら処理を進めているはずです．したがって，ある回路ブロックはほかの回路ブロックが特定の動作を行うという仮定のもとで自己の動作を決めています．この仮定しているほかの回路ブロックの動作以外の動作はドント・ケアということになります．

このドント・ケアを利用して回路最適化を行っている場合，等価性検証の際にもそのドント・ケアに相当する条件を順序回路向けの等価性検証ツールに指示してやる必要があります．この指定がけっこう複雑だったり，煩雑だったりします．

また，指定条件がゆる過ぎると，等価な設計が「不等価である」と判定されます．逆に，指定条件がきつ過ぎると，不等価な設計を「等価である」と判定する可能性があります．この点については，個々の設計プロジェクトごとにさまざまな形でノウハウを蓄積することが重要になります．

ふじた・まさひろ
東京大学 大規模集積システム設計教育研究センター（VDEC）教授

<筆者プロフィール>
藤田昌宏．1985年，東京大学大学院 情報工学専門課程修了．工学博士．同年，富士通研究所入社．論理設計用CAD技術の研究開発に従事．1993年より米国富士通研究所へ出向，CAD研究グループを立ち上げる．2000年3月より東京大学大学院 工学系研究科電子工学専攻 教授．2004年4月より東京大学 大規模集積システム設計教育研究センター（VDEC）教授．ディジタル・システム設計支援技術，とくにシステム・レベル合成，論理合成，論理検証を中心に研究している．